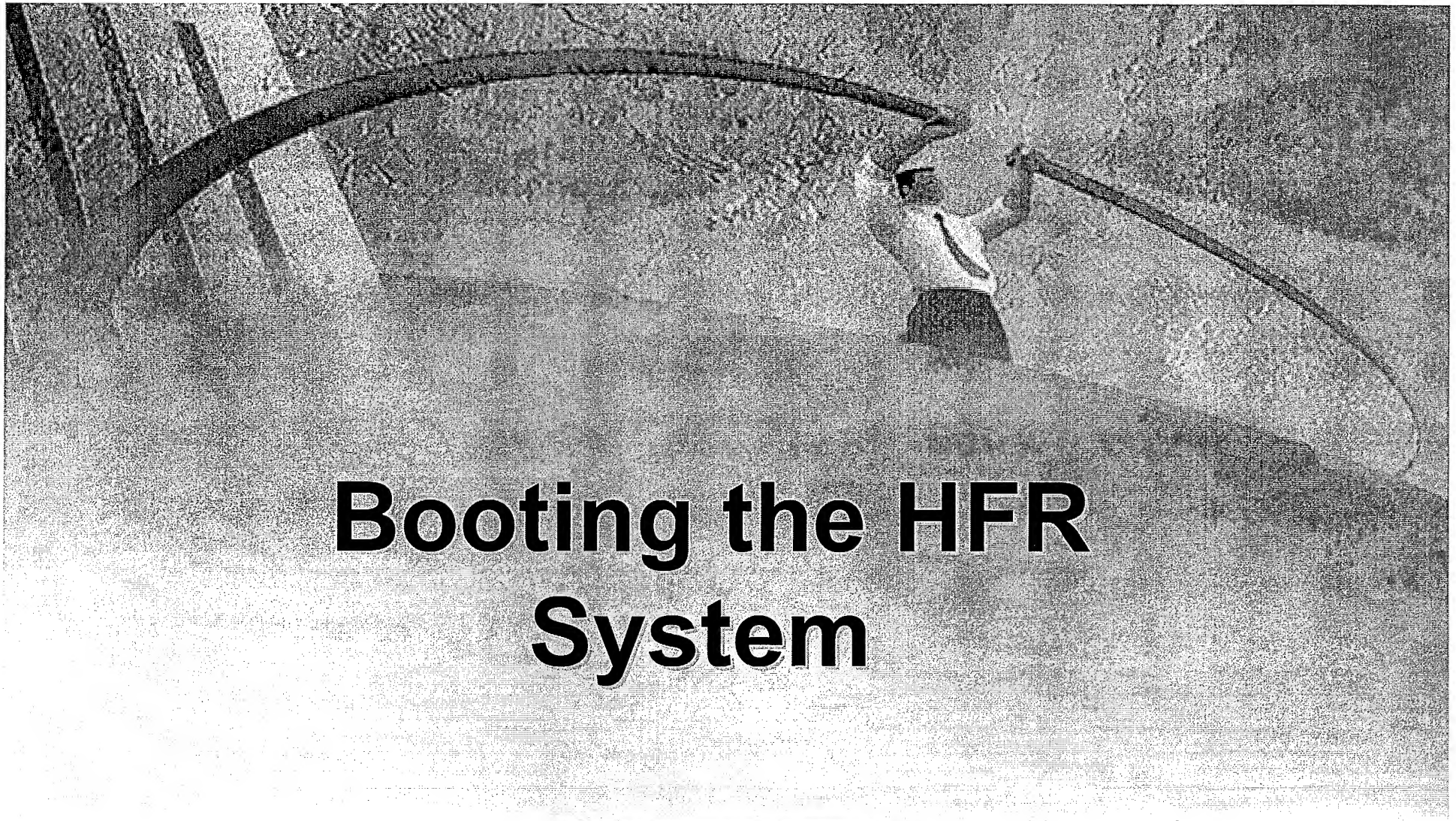


Booting the HFR System

[redacted]

Doug Wooff

Cisco.com

EDCS-305725

EXHIBIT 1.1

Contents

Caveats for this Boot Discussion

- Install suite is currently LR-unaware
- Assume certain boot configs already set (rack serial numbers, etc)
- Details of diskbooting glossed over
- Details of boot time improvements coming in future presentation

HFR Software Delivery Model

The HFR Software Delivery Model

- Initial composite "vm" (bootable image) is delivered and booted into dSC memory
- End users install image to disk on dSC, reboot
- Developers may skip install to disk
- All subsequent non-dSC nodes boot from dSC, both management and managed nodes

The HFR Software Delivery Model - 2

- Non-dSC nodes will follow "boot model" of dSC
- Means that if dSC is membooted, all other nodes will be membooted
- Memboot is model for developers only, is non-persistent
- If dSC is diskbooted, other nodes will cache their MBI's and packages, ready for speedy second boot

The HFR Software Delivery Model - 3

- Non-dSC management nodes will always sync *all* packages locally from dSC
- Means that any management node is capable of becoming dSC
- Initial software distribution stabilizes using only the one original composite previously installed on dSC
- All following software delivery is by PIE

Right Way to Boot HFR Systems

Right Way – Is there any other way?

- The "Right Way" is the easiest way
- Keep all RP's dormant (in rommon or unpowered)
- Bring up the one dSC node first
- Reset any other RP to rommon, let it autoboot
- System takes care of itself
- EASY!

Right Way – 2

- Install management nodes (RP/SC) will time out and boot whatever's on their disks – in absence of dSC
- This is a problem if you let "random" incompatible sets of packages (ie. composites) boot on various RP's
- Random image booting will be *corrected (*as much as possible*)*
- Wrong (ie. unknown or incompatible) software will not be permitted to persist

Right Way – Summary

- Boot the dSC first
- Boot all other RP's (standby, other racks) using rommon to autoboot
- Managed nodes (SP, LC, DRP) will be booted by shelfmgr, mbimgr, and instdir automatically (if rommon set to autoboot)
- Right way is easiest, least error-prone, uses rommon and MBI

Other Ways to Boot HFR Systems

Other Ways – Is this a wrong way?

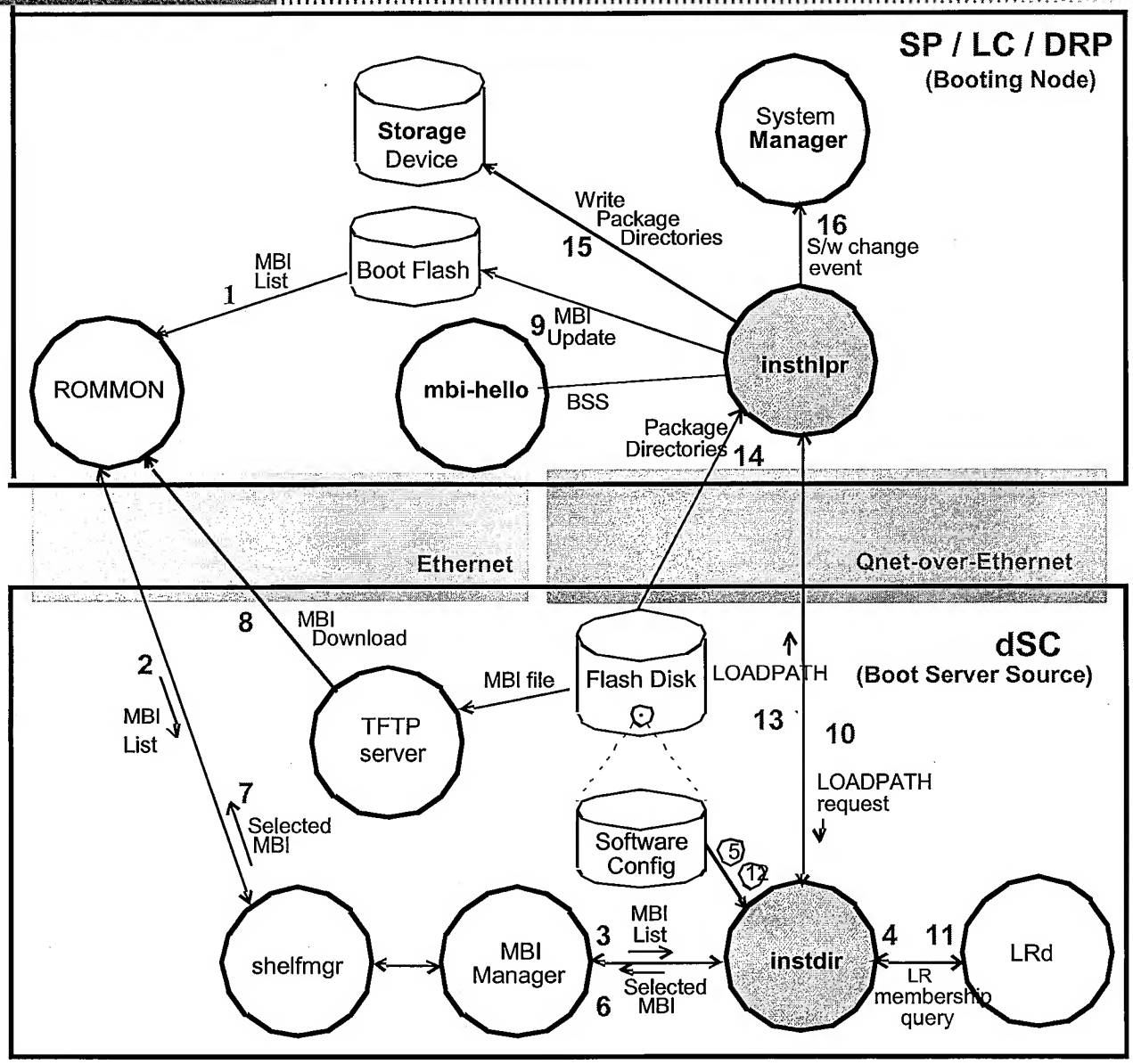
- If "Right Way" is the easiest way, why would anyone do something else?
- Some developers like to boot a *different* second composite on a second rack
- This will soon be disallowed, and *corrected* (if possible)
- Booting a second *identical* composite will be allowed – as a convenience - on standby RP or 2nd rack – but is not necessary

Typical

"Managed"

Node Boot Sequence:

16-step Program


SP / LC / DRP
(Booting Node)
Storage Device
System Manager
Write Package Directories
15
16
S/w change event
Boot Flash
MBI List
1
ROMMON
9
MBI Update
mbi-hello
BSS
insthlpr
Package Directories
14
Ethernet
Qnet-over-Ethernet
8
MBI Download
2
MBI List
TFTP server
MBI file
Flash Disk
LOADPATH
dSC
(Boot Server Source)
13
10
LOADPATH request
7
Selected MBI
Software Config
5
12
shelfmgr
MBI Manager
3
MBI List
6
Selected MBI
instdir
4
11
LRd
LR membership query

Managed Node Booting Sequence

1. Reset to rommon, looks for local MBI's in flash
2. Rommon sends out list, which can be empty, in a BOOT_REQUEST msg
3. Shelfmgr receives BOOT_REQUEST msg, passes it to mbimgr
4. [FUTURE]: consult with LRd
5. mbimgr/instdir mbimgr decide which MBI should be running
6. Return MBI filespec to shelfmgr
7. Return MBI filespec to rommon
8. Rommon boots local MBI if indicated, or initiates TFTP d/l

Managed Node Booting Sequence – cont'd

9. insthelper comes to life inside MBI, gets BSS from mbi-hello

10. insthelper locates instdir, requests LOADPATH

11. [FUTURE]: consult with LRd

12. Using s/w config, instdir looks up LOADPATH for node

13. instdir sends LOADPATH to insthelper

14. insthelper updates MBI in flash if needed

15. insthelper pulls down correct packages in "bundles", writes pkgs to local storage (memory or disk)

16. insthelper sends s/w change notification to sysmgr/sysmgr_lite

"Management"

Nodes:

They are

Different

Management Nodes

- Only one Primary Install Director (pID) in entire system
- pID coordinates *all* bootup (initial package pull) and subsequent software upgrade activities for all nodes
- pID executes only on the dSC, is sleeping on all other management nodes
- By definition, any management node is capable of becoming dSC (ie. RP or SC nodes)
- By extension, any node capable of becoming dSC must be ready to host pID
- All possible software for all nodes must be sync'd to all management nodes; must replicate pID view

Management

Node Boot Scenarios

Single Standalone RP

- Boot composite.vm from rommon
- Only MBI portion of composite executes
- sysmgr_lite (aka 'init') runs, starts .init processes
- insthelper (I-H) runs, pauses to look for pID
- sysmgr_lite will *not* proceed until prodded by I-H
- insthelper times out, updates loadpath to reflect entire composite, pulses sysmgr_lite to start sysmgr
- Entire composite now boots
- Same behaviour for netboot or diskboot

Two RP's Booting - Simple

- Boot composite.vm from RP1 rommon, RP1 goes dSC
- Boot composite.vm from RP2 rommon
- Only MBI portion of RP2 composite executes
- RP2: sysmgr_lite (aka 'init') runs, starts .init processes
- RP2: insthelper (I-H) runs, pauses to look for pID
- RP2: insthelper finds pID, validates expected loadpath against packages "in stock" (in rest of composite)
- RP2: packages validated, I-H pulses sysmgr_lite to start sysmgr and rest of composite
- RP2: entire composite now boots
- This behaviour only when both RP's netboot or diskboot

Two RP's Booting – Sync Needed

- Boot composite.vm from RP1 rommon, RP1 goes dSC
- Boot composite.vm from RP2 rommon
- Only MBI portion of RP2 composite executes
- RP2: sysmgr_lite (aka 'init') runs, starts .init processes
- RP2: insthelper (I-H) runs, pauses to look for pID
- RP2: insthelper finds pID, validates expected loadpath against packages "in stock" (in rest of composite)
- RP2: packages INVALID, I-H syncs correct packages
- RP2: tell sysmgr to start correct packages
- RP2: booting completed
- This behaviour only when both RP's netboot or diskboot

Two RP's Booting – Disk Sync Needed

- Diskboot from RP1 rommon, RP1 goes dSC
- Netboot composite.vm from RP2 rommon
- Only MBI portion of RP2 composite executes
- RP2: sysmgr_lite (aka 'init') runs, starts .init processes
- RP2: insthelper (I-H) runs, pauses to look for pID
- RP2: insthelper finds pID, validates expected loadpath against packages "in stock" (in rest of composite)
- RP2: packages INVALID, I-H syncs correct packages
- RP2: I-H sets BOOT var + autoboot confreg bit
- RP2: self-reset, autoboots from disk after reset
- RP2: boot from disk completes, sync is NOP this time

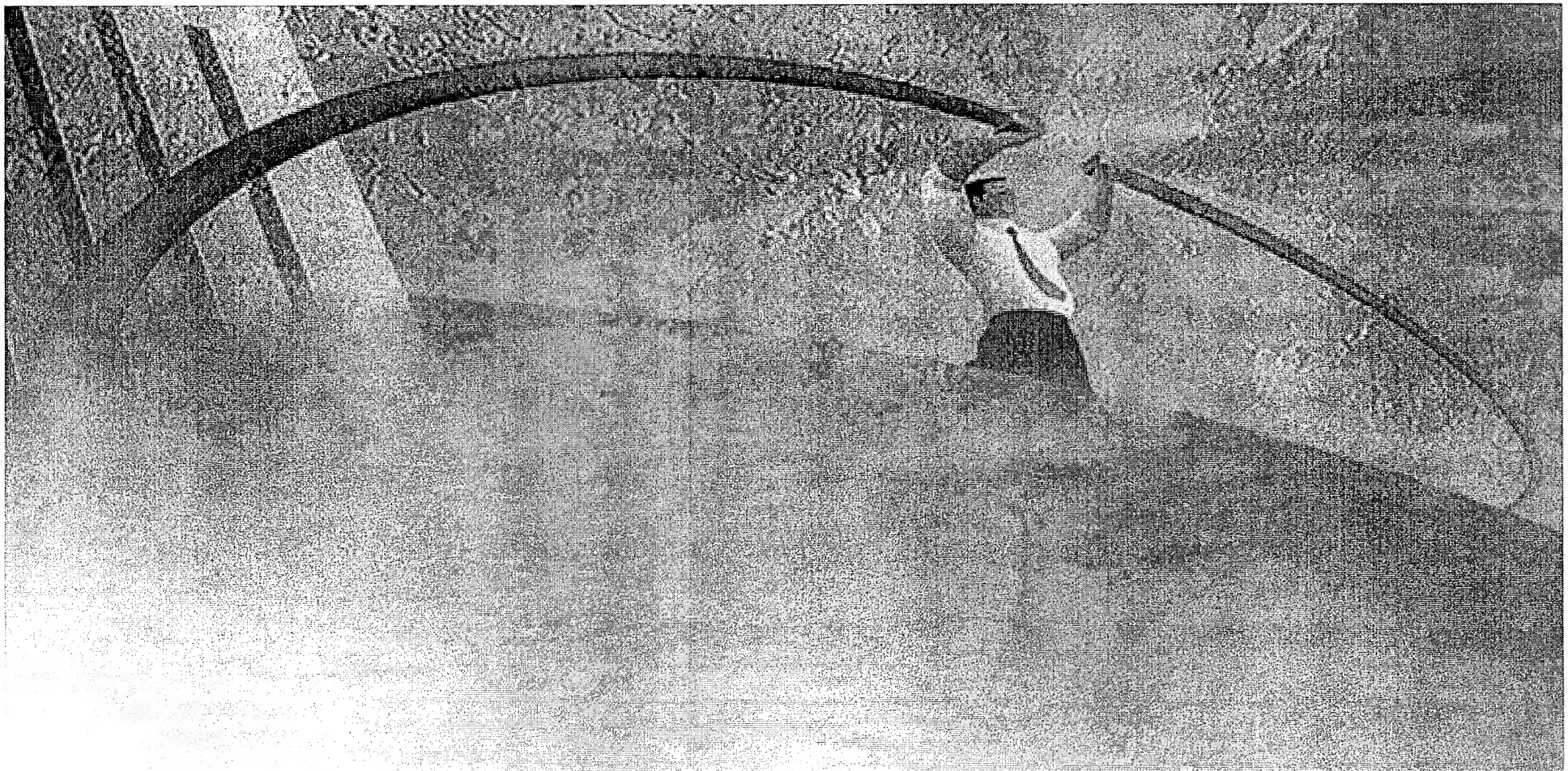
How GSR Fits into this model

How does GSR fit in?

- The precise "Right Way" (like HFR) is not available on GSR hardware
- GSR rommons cannot "autoboot", fetch an MBI – has no tftp client
- There is no standalone GSR RP MBI
- Want booting to be similar to HFR for support and maintenance

How does GSR fit in? - 2

- Can bring up first RP (becomes dSC) just like HFR, either memboot or diskboot
- Second RP – standby RP – can likewise be booted using a second composite
- Booting the second composite on standby will automatically sync to the first RP's packages
- Guarantees proper software distribution



Appendix

What is MBI - really?

- Minimum Boot Image (not Mysterious)
- Bare essentials to create a “node”
- Node is visible via IPC of choice
- QNET == IPC of choice
- QNET is essential, required
- Node “floats” until gets app code
- MBI node has no assumed role (active/standby/dSC)

So what's the deal with MBI?

- Enables Packaging for SW delivery
- Makes application upgrades easy
- Contains critical, slow-changing core
- Same as "OS Package"; MBI is just a name
- Key application is insthelper
- Is part of the HA story

References

EDCS-194266 HFR Boot Arch FRD
EDCS-201202 HFR Boot Arch FS
EDCS-136393 HFR MBI Manager

EDCS-102416 CCP